Australian Government
Department of Defence
Defence Science and
Technology Organisation

Review of Software Platforms for Agent Based Models

Matthew Berryman

Land Operations Division
Defence Science and Technology Organisation

DSTO-GD-0532

ABSTRACT

This report reviews both general agent based modelling (ABM) and battlefield-specific ABM toolkits against established criteria, namely flexibility, documentation, speed, and facilities. The other main focus of this report is to consider which ABM toolkits are best suited to studying self-organisation, adaptation, and causality in networks, since these are all essential parts of complex adaptive systems of interest to defence. The ABM toolkits evaluated against these requirements were BactoWars, EINSTein, MANA, MASON, NetLogo, Repast, Swarm and WISDOM-II.

RELEASE LIMITATION

Approved for public release

Published by

Land Operations Division
DSTO Defence Science and Technology Organisation
PO Box 1500
Edinburgh South Australia 5111 Australia

Telephone: (08) 8259 5555
Fax: (08) 8259 6567

AR 014-164

Submitted: January 2008
Published: April 2008

APPROVED FOR PUBLIC RELEASE

Review of Software Platforms for Agent Based Models

Executive Summary

Many complex systems are amenable to study through the use of agent based models. Agent based models provide useful insights into how individual interactions give rise to emergent properties. Of these, structures and causality are of great interest.

The focus of this report is firstly to evaluate both general agent based modelling (ABM) and battlefield-specific agent based modelling (BSABM) toolkits against established criteria: speed, flexibility, documentation provided, and general facilities (for example graphing facilities) provided. The toolkits are also assessed for statistical capabilities in order to highlight this area as needing (generally) better support from ABM toolkits.

The other main focus of this report is to consider which ABM toolkits are best suited to studying structure formation and causality in networks. Two processes that generate and/or preserve structure—adaptation and self-organisation—are considered and the toolkits are evaluated as to how well they support these two processes. The toolkits are also evaluated as to what facilities (if any) they have for supporting causal decision making in networks of agents. These three criteria together are critical for modelling complex adaptive systems, of which battlefield scenarios are one example of importance to defence. A simple ABM was developed to conduct the evaluation of the ABM platforms for their support of adaptation, self-organisation, and networked causality, along with more general criteria established in previous ABM platform reviews.

MASON is the overall best choice, and fit for the purpose of exploring complex adaptive systems, but requires a high level of programming experience. For simple models, and for people with little programming experience, NetLogo is highly recommended. EINSTein is recommended as a BSABM platform requiring no programming experience.

Contents

Glossary **vii**

1 Introduction **1**

2 Requirements **2**

3 Methodology **4**

3.1 Overview 4

3.2 A Test Scenario: SimpleArmy 6

4 Results and Discussion **7**

4.1 Raw scores of the evaluation 7

4.2 Summary and interpretation of results 8

4.3 Comments on specific ABM toolkits 9

4.3.1 BactoWars 9

4.3.2 EINSTein 10

4.3.3 MANA 10

4.3.4 MASON 11

4.3.5 NetLogo 11

4.3.6 Repast 12

4.3.7 Swarm 13

4.3.8 WISDOM-II 13

4.4 Previous analyses 14

4.5 Comparison with previous analyses 15

5 Conclusions and Future Work **15**

A Software Code **16**

A.1 Java **16**

A.2 NetLogo **16**

A.3 Objective C **17**

References **18**

Glossary

ABM Agent Based Model

BSABM Battfield-Specific Agent Based Model

C2 Command and Control

C3 Command, Control and Communications

CS Complex Systems

CAS Complex Adaptive Systems

EINSTein Enhanced ISAAC Neural Simulator Toolkit, where ISAAC is Irreducible Semi-Autonomous Adaptive Combat.

MANA Map-Aware Non-uniform Automata

NCW Network Centric Warfare.

Repast REcursive Porous Agent Simulation Toolkit

WISDOM-II Warfare Intelligent System for Dynamic Optimization of Missions, version two.

1 Introduction

Agent based modelling is a useful tool in simulating and exploring phenomena that consist of or can be thought of in terms of interactions between individual agents, for example, people in a football stadium or ants foraging for food. Typically the systems of interest are complex systems (CS) or complex adaptive systems (CAS) (see below). This report assesses the capabilities of Agent Based Modelling (ABM) toolkits (Table 1), including battlefield-specific ABM (BSABM) platforms[1], against requirements for analysing self-organisation, adaptation, and networked causality. A simple ABM was developed to help evaluate ABM platforms for these criteria, which are essential for studying CS and CAS.

Table 1: The platforms reviewed in this report, whether they are general or battlefield-specific, and, if relevant, the programming languages used in conjunction with the platform.

Platform	Subsection	General / Battlefield-specific	Programming Language(s)
BactoWars	4.3.1	Battlefield	Java
EINSTein	4.3.2	Battlefield	Python (optional, for batch runs)
MANA	4.3.3	Battlefield	N/A
MASON	4.3.4	General	Java
NetLogo	4.3.5	General	Logo-variant
Repast	4.3.6	General	Java
Swarm	4.3.7	General	ObjC or Java
WISDOM-II	4.3.8	Battlefield	N/A

Complex systems (CS) are characterised by emergent, non-linear behaviour – they are dynamic open systems in which the behaviour of the whole is not just a simple sum of the parts. This emergence may be weak emergence, in which we could theoretically reduce the behaviour into the sum of the parts, but this may be too difficult and there exists a simpler, more efficient rule for the overall behaviour. It may also be strong emergence, in which the behaviour of the whole is irreducible into the behaviour of the component parts considered in isolation or in other combinations.

Complex adaptive systems (CAS) are complex systems in which adaptation occurs—that is, they change in response to their environment (which may include CAS, such as adaptive agents). This can occur at one or more levels of a complex system. For example, this may take the form of selection, in which agents, or groups of agents, pass on their traits with a higher frequency than agents that are less fit according to some measure. Agents can also learn, given feedback on their performance, and hence modify their behaviour. Battlefield scenarios are thus typical of complex adaptive systems in that the include a large number of agents learning and acting in dynamic ways that give rise to emergent behaviour such as success or failure of forces.

ABMs have been used in a wide variety of fields. Athale, Mansury & Deisboeck [2005] used Repast to study tumour growth, using a multi-scale, multi-cellular CAS ABM. Athale

[1]For military applications, ABMs have historically been called agent based distillations (ABDs). However the use of this term to differentiate from general ABMs is no longer accurate, since many battlefield models are not "distilled" but are in fact highly detailed. Most BSABMs provide a fixed set of movement, attack, enemy detection and other algorithms; however BactoWars is customisable and its algorithms can be modified or replaced.

& Deisboeck [2006] took the same model and used it to look at a specific cell receptor. ABMs have extensive use in modelling CAS in economics [Tesfatsion 2003]. Other areas where ABMs are commonly used to model CAS are in the social sciences [Epstein & Axtell 1996] and ecology [Grimm et al. 2005].

There have been a number of other reviews of ABM platforms. Railsback, Lytinen & Jackson [2006] wrote a review of agent based model platforms, focussing on Repast, Swarm, MASON, and NetLogo. The review by Najlis, Janssen & Parker [2001] provides a good summary of capabilities of Swarm, Repast, CORMAS and another ABM platform not reviewed here, namely Ascape, which is no longer in development. These two review papers are discussed later, in order to elucidate a list of general ABM requirements in addition to those for structure formation, adaptation and networked causality. This report extends on and updates these previous reviews, with a focus on ABM toolkits for use for CS and CAS.

The following section gives some necessary background and definitions on self-organisation, adaptation, and networked causality.

2 Requirements

Newth & Finnigan [2006] define self-organisation as meaning "that the system, properly defined, can increase its internal order over time without the imposition of order (information) from outside the system boundary." They then go on to point out this implies a decrease in system entropy, so therefore self-organizing systems must not be closed. Ryan [2007] takes this point further to define self-organisation as "a process that increases the organisation of a system away from equilibrium, in contrast to self-assembly", which "increases the organisation of a system by moving towards equilibrium."

To think about structure more generally, one can consider at the most basic level of an ABM, that structure which arises through basic agent-agent interactions. Examples of this include Conway's Life [Gardner 1970] and flocking "boids"[Reynolds 1987]. At a higher level, one can also consider interactions between agents and their environment. Real-world structures exhibit interaction within and between different levels—consider the economy, at the microeconomic scale there are individual people and firms buying and trading, and at the aggregate macroeconomic level there are different consumer price indices and interest rates, and these then feed back into actions, and hence structures, at the microscale. The way information is used by agents at different levels in making decisions provides insight into how structures and patterns form.

Simple agent-agent interactions, including stigmergy, are the lowest level of networked causality. Stigmergy is indirect communication using the environment [Theraulaz & Bonbeau 1999], for example, ant communication through pheremone trails. In these simple agent-agent interactions, only one piece of information is acted upon. At a higher level, one can consider a linear combination of influences including the agents' own internal values on decisions. This is of course often done in a probabilistic fashion, in the presence of noise. At a still higher level, agents may act in a non-linear combination of influences, where two variables may combine to strengthen or act in opposition in decision making. An example of non-linear networked causality might be if one receives two email invites to

lunch, one from a friend and one from a supervisor, and the latter takes precedence over the first one.

In order to transition from studying complex systems, to complex *adaptive* systems, it is important to have facilities available for learning in individual agents, or selection occurring at an agent or group level. Adaptation can be generally defined as occurring whenever there is some sort of variation, and feedback (either directly or via proxy measures) on the variation, which allows for selection of "fit" (or merely "good enough") agents / strategies. This tendency for selection produces agents that then more closely match their environment (at least in part), on average. Two particular instantiations of adaptation are evolution and learning.

Based on the background and definitions above, the key features required to support adaptation are

1. A means of variation of agents, as a group, individually, or both.
2. That the variation of agents should lead to sets of useful actions, that is, ones that produce ordered randomness as opposed to randomness or strict order.

For adaptation to occur, there must also be mechanisms for reproduction, or at least repetition of a strategy, and mechanisms for feedback. A further requirement for the development of structure, or self-organisation, is that agents interact with each other, between themselves, with their environment, or even across multiple levels of aggregation. Different types of decision making based on communication between agents support the study of networked causality (and how this plays a role in self-organisation).

In summary, the important general issues identified by Najlis, Janssen & Parker [2001] and Railsback, Lytinen & Jackson [2006] for ABM platforms are:

Open-source is the ability to read and update the instructions that comprise the platform. This can be of importance where there are bugs in the platform, or if the documentation is poor. This can also aid in extending the facilities provided by the software, or adding in other platforms to provide missing facilities.

Flexibility is the ability to write custom agents and agent behaviours.

Speed of execution is important, particularly under statistical replication and also when a variety of scenarios or parameters need to be explored.

Documentation is important in order to fully use the platform, or for those ABM platforms that consist of libraries, to even start to use the software. The support provided by user communities can also be of high importance and this is considered along with documentation—this support is often archived online for future reference.

Facilities What facilities does the ABM platform have for drawing graphs, recording simulation data to file for further analysis, etc.

The role of statistics in ABMs is important, not for predicting quantitative behaviours in the system being modelled, but rather for quantitative analysis of the behaviour of the

model. While qualitative outputs—in particular graphical displays–are useful for ascertaining qualitative behaviour of the models (and also in conference presentations), it is the quantitative analysis of the model that provides substantive insights. Often simple statistics are not fully descriptive of emergent behaviours. It may be that the red team has a higher average score than a differently-structured blue team, but this may also have a higher variance, indicating that for some choice of parameters it performs much worse than the blue team. Similarly, basic network statistics may indicate that a graph is random, but some of these random graphs may have different degree distributions arising from emergent behaviour of the system. It is with cases like this in mind that the statistical tools available in the ABM and BSABM platforms were evaluated herein as distinct from other facilities provided.

3 Methodology

3.1 Overview

Each ABM platform was assessed against the categories identified by other reports and fellow modellers as being important to ABMs, namely: flexibility, documentation, speed, and facilities. Statistical and other forms of analysis, such as machine reasoning about agent behaviours, were considered separately from other facilities such as graphing. This is because analysis is an important area in assessing ABMs for use in modelling. The areas of adaptation, structure formation, and networked causality were also assessed. Since these are important requirements for using ABM and BSABM platforms in studying CS and CAS, they were given a weighting of two (instead of one for the rest) when calculating a final score. A rating scheme was devised, and this is given in Table 2. This was based on discussions of experiences of ABM platforms with other ABM experts, however could be further refined in future work. In some categories it was easy to make an objective assessment, for example platforms either had support for evolution or they did not. In others such as documentation and user community support the assessment was partly subjective. In future work, a refined set of criteria will be established and assessments by a group of active users will be canvassed to make this more rigourous.

A simple ABM (SimpleArmy, documented below) was used in conjunction with sample code / scenarios to study the features of each platform, and in the case of general ABM platforms an examination of the API (application programming interface, the documentation on the libraries) was used as well. In the case of generic ABM platforms where code can be written, ease of implementation of adaptation, mechanisms of structure formation, and networked causality were considered. In short, this came down to whether the API had specific facilities for supporting these features. Obviously, with some extra work, this functionality could be imported from other libraries (for example using ECJ with Repast) or could be written by the user.

Table 2: Table showing the definitions of poor, good, and excellent for the categories in which the ABM platforms were assessed. Scores as used in Tables 3 and 4 are also listed in column headings. More advanced features (adaptation, structure, and causality) were either present or not present in the BSABMs, for the more flexible, programming-based general ABM platforms, library/language support for those features was considered.

Area	Poor (score 1)	Good (2)	Excellent (3)
Flexibility	Single algorithm / few algorithms with variable parameters	Multiple algorithms with variable parameters	User-specified algorithms
Documentation	Terse/non-existent examples, many methods/functions not documented	Reasonable examples, Most methods/functions documented	Useful examples, most methods/functions documented, helpful user community (mailing list / web forum)
Facilities	Limited data logging	Basic data logging	Extensible data logging and graphing
Analysis	Computes averages of some variables	Supports basic statistical methods (eg. mean, standard deviation, etc.)	Supports advance statistical methods (e.g. distribution-type checking)
Speed	Noticeable time (several seconds) for screen refreshes	Runs with occasional pauses / slows down with ≥ 20 agents	Runs well on screen / has batch mode
Adaptation	Merely reactive	Evolution of agent attributes	Evolution of agent algorithms, including learning
Self-organisation	Basic swarming capability of agents (feedback between agents)	Feedback between agents and their environment and other agents	Multi-level feedback
Networked causality	Stigmergy / Basic communication of one item of causal influence	Agents decide based on a linear combination (weighted sum) of information	Agents are influenced in a nonlinear fashion based on information received

3.2 A Test Scenario: SimpleArmy

SimpleArmy was designed to represent a simple battlefield scenario, in order to test general facilities of the ABM and BSABM platforms, as well as their facilities for self-organisation, adaptation and networked causality. It was designed in order to provide for a good comparison of both general ABM platforms and BSABM platforms in terms of a battlefield scenarios. The model was designed to show complex adaptive systems behaviour, in particular emergence, by focussing on agent-agent interactions in the battlefield and over a network. Most of the notes on the ABM platforms in the following section were written based on experiences in implementing SimpleArmy in the different ABM and BSABM platforms. These experiences also helped in evaluating the ABM platforms against the criteria in Table 2. In implementing SimpleArmy in the BSABM platforms, the existing BSABM movement and communication algorithms (where available) were used. Evolution was not included in the model, however an assessment was made on this feature if provided by the platforms reviewed. This was done by reviewing the documentation and examining existing ABMs written using those platforms. Only a simple model of learning was implemented. The ABM is also not meant to be a detailed, or particularly accurate representation of real-world battles, since that was not the purpose of this report. Code for Repast, MASON, and NetLogo implementations is available on request. For the full set of code, please contact the author. The following is a brief summary of facets of the SimpleArmy model.

The SimpleArmy battlefield has the following structures:

- A terrain, which simply provides an elevation z_o at locations (x_o, y_o).
- A ground space, which contains a maximum of one ground agent at each (x, y) location.
- An air space, which contains any number of air agents, since these can be at different altitudes.
- A communications network, which connects members of each team to some members (all members in a fully-connected network, or no connections if the network is fully disabled) of the same team.
- The actual agents, which are the members of friendly (blue) or enemy (red) teams, and consist of a maximum of one headquarters per team, and any number of fixed platform surface-surface missiles (SSMs), tanks, and attack helicopters. Civilian agents (yellow) were not considered.

The distinction between ground and air spaces, as well as terrain in the ground space, provides for diversity of agents and also in diversity of possible interactions; this is important for modelling complexity.

Agents can perform any one of a number of different actions:

- *look*
  This searches the Moore neighbourhood within sight range of the agent and updates the list of enemy agents.

- *removeDeadEnemies()*
  This goes through the list of enemy agents, and removes those with $hp \leq 0$ (that is, those that are dead).

- *move()*
  This moves the enemy in the following way:

  A target location is selected either from an order (if one exists), otherwise the highest scoring enemy in the agent's list of enemies is selected.

  1. *target(boolean targetEnemies)*
     If the parameter *targetEnemies* is false, this also includes friendly agents in the search. If an order exists, and it is to *PURSUE*, then the agent will pursue but not attack that agent, if it is set to *ATTACK* then it is
  2. *sendmsg()*
     This sends a message out on the network to some subset of other friendly agents on the same team, about a particular enemy agent.
  3. *recvmsg()*
     This takes an message (a copy of information about an enemy agent) and either adds it to the list of current enemies, if no information on that enemy exists, or updates the entry corresponding to that enemy in the list of enemies.

Each agent has a number of properties, including sight range, movement range, firepower, and firepower range, which define the specific behaviour of each class of agents. Each agent has a number of actions per time step, the specific actions from the above list performed depends on the agent type, for example headquarters cannot move, so do not call *move()* and instead can send orders. The *look()* method is called before *removeDeadEnemies()* is used, as it requires an up-to-date list of enemies on which to operate, including ones recently killed on the battlefield.

The diversity of agents, as well as the number of possible interactions between them, particularly network interactions, helps make the model complex. It also helps to give rise to emergent behaviour, in the sense that there are global behaviours which are difficult to predict from the local interaction rules.

4 Results and Discussion

4.1 Raw scores of the evaluation

The following consists of a set of notes on the various ABM and BSABM platforms reviewed, based primarily on experiences in implementing SimpleArmy, in the case of the BSABMs this mainly consisted of parameter tuning rather than implementing the algorithms described above from scratch. Quantitative and qualitative assessments were made according to the criteria in Table 2. These are given in Tables 3 and 4, and the results (apart from the yes/no open-source column) are on a scale of 0-3, 0 being "nonexistent", 1 meaning "poor", 2 meaning "good" and 3 meaning excellent. In Table 3, open source means the source code can be read and modified. Flexibility is in terms of ability to add

Table 3: Table comparing the ABMs and BSABMs reviewed against general criteria. Scores are as per Table 2.

Software	Open-source	Flexibility	Speed	Documentation	Facilities	Analysis
BactoWars	Yes	3	2	2	2	1
EINSTein	No	1	3	2	3	2
MANA	No	1	2	3	1	1
MASON	Yes	3	3	2	3	2
NetLogo	No	3	3	3	3	2
Repast	Yes	3	3	3	3	2
Swarm	Yes	3	2	2	3	2
WISDOM-II	No	1	1	1	2	2

Table 4: Table comparing the ABMs and BSABMs reviewed against specific criteria to enable modelling of CS and CAS. Scores are as per Table 2.

Software	Adaptation	Self-organisation	Networked causality
BactoWars	1	2	3
EINSTein	1	3	2
MANA	1	1	0
MASON	3	2	3
NetLogo	2	3	1
Repast	2	2	3
Swarm	1	3	1
WISDOM-II	2	3	3

any new features desired in a model. Speed was not quantitatively assessed apart from a stopwatch time on the SimpleArmy model, and is difficult to compare between the general ABMs platforms and BSABMs as the SimpleArmy model could not fully be implemented in the BSABMs. Documentation includes user communities and the level of support they provide. Facilities includes visualisation support, but also other facilities provided such as data logging and parameter file reading support. Analysis includes statistical packages but also things like the reasoning ability in WISDOM-II and the ability to inspect agent attributes and behaviours on-screen in the general purpose ABMs. Note again that in the case of ABM platforms where code can be written, ease of implementation of adaptation, mechanisms of self-organisation, and networked causality were considered; in short, this came down to whether the API had specific facilities for supporting this. Obviously this functionality could come from other libraries, for example using ECJ with Repast, or could be written by the user.

4.2 Summary and interpretation of results

Table 5 shows the final scores (out of a maximum of 33) for each of the platforms. The score was calculated as the sum of the scores in Table 3 plus twice the sum of the scores in Table 4 (that is, the subtotal from Table 4 was given a weighting of two), since these are more important criteria when considering ABM and BSABM platforms for modelling CS and CAS. Within each category—general usability criteria in Table 3, and CS and CAS criteria in Table 4—the results were given equal weighting as they are approximately equal in importance. MASON and Repast are clear leaders on the combined score of standard

Table 5: Subtotals and total score for each ABM platform reviewed. The maximum score possible is 33.

Software platform	usability	applicability to CS & CAS	appliciability × weighting	Total score
BactoWars	10	6	12	22
EINSTein	11	6	12	23
MANA	8	2	4	12
MASON	13	8	16	29
NetLogo	14	6	12	26
Repast	14	7	14	28
Swarm	12	5	10	22
WISDOM-II	7	8	16	23

criteria and criteria of interest to modellers of CS and CAS. NetLogo would have scored as highly if not for its lack of facilities for exploring networked causality. The low score of flexibility for EINSTein may be a "show stopper" if a general ABM must be developed, otherwise its high score indicates it would be of use in exploring CS and CAS concepts in the battlefield scenario it models. MANA's low score indicates that it is not a good BSABM for general use or for exploring CS and CAS concepts. The following section goes into more detail about each of the toolkits.

4.3 Comments on specific ABM toolkits

4.3.1 BactoWars

BactoWars is a DSTO-built, Java-based ABM platform for exploring combat models. It supports loading resources (icons, maps, behaviours) from JAR and ZIP files. A BactoWars scenario consists of icons and behaviours, selected from the file, and a set of attributes for the classes of agents. These classes of agents can then be placed on a map at specified locations, and the simulation can then be run.

BactoWars comes with a large number of pre-built modules for everything from networking agents together, to swarming behaviours, to targeting and logging of output. However, many of these behaviours are buggy, and documentation on how they interact (for example, network initialisation and network use) or are to be used is missing. New behaviours are easy to write provided the user has some programming experience, and behaviour templates are generated by the software. In BactoWars, it is easy to load new maps, custom icons, and behaviours. BactoWars supports loading these from a web-distributed JAR or ZIP file. Of all the software packages reviewed, BactoWars alone comes supplied with a way of logging software errors, although this could easily be handled in other ABMs by using an Integrated Development Environment (IDE) capable of logging these and even linking them back to specific lines of software containing bugs. The interface of BactoWars is generally easy to use but is slightly counterintuitive in areas and can be slow at times. Like MASON, BactoWars can generate movies of simulations, but at a lower quality compared with MASON-generated movies. There is a limited user-community.

BactoWars is recommended when extra functionality is required to be added, but a BSABM framework is still required.

4.3.2 EINSTein

EINSTein is the The Enhanced ISAAC Neural Simulator Toolkit, where ISAAC is "Irreducible Semi-Autonomous Adaptive Combat", an earlier BSABM with similar features but a difficult interface [Ilachinski 1999]. Both products originate from the Center for Naval Analyses. It is a BSABM (battlefield-specific ABM), and has many features suitable for exploring CS and CAS in battlefield scenarios.

Of the BSABMs tested, EINSTein is the fastest, and it is easy to batch multiple runs over a parameter space through the use of relatively simple Python scripts. It also has built-in functions for parameter sweeps, and can plot the resulting fitness landscape according to battle metrics. Since it is geared towards exploring CAS, it features built-in neural network, pattern recognition, and genetic algorithms. It also has built-in algorithms for measuring statistics. It is capable of modelling network centric warfare (NCW) concepts through the network facilities provided in the software, though these are restricted to implementing linear causality. All of the algorithms are fixed, and while they can be modified slightly through parameters, cannot be changed in the way they operate or be added to. EINSTein has support for hierarchies at the battle-group level. This facilitates structure formation, by enabling multi-level feedback between agents and groups of agents. There is no support for true adaptation other than merely reaction of agents to changes in their environment. On the negative side, it is Windows specific, only one squad is highlight-able at a time (making it hard to identify behaviour at the squad-squad level), and it suffers from "parameter bloat". At best, it is hard to specify a simulation as a large number of parameters are required to be input. It is possible some of these are not available, or irrelevant to a scenario being explored. If the model is to have any predictive power, then having too many parameters can lead to over-fitting of the model, where it too closely represents a specific scenario. To quote Castle & Crooks [2006]: "Predictive models of this nature can be insufficiently general to represent a large range of potential outcomes related to the system under analysis, or to analyse alternative scenarios". Nonetheless, EINSTein is a useful and fast package for BSABMs.

While it is not capable of being extended, and suffers from a few minor flaws, including the possibility of "over-fitting" through a large number of parameters, EINSTein is nonetheless very capable, fast software, with a lot of features useful for modelling CS and CAS in battlefield scenarios.

4.3.3 MANA

MANA is the Map-Aware Non-uniform Automata simulation, developed by the New Zealand Defence Technology Agency. It is a well-established BSABM and has had many models developed in it. It is good at what it does, but lacks features useful in modelling CS and CAS in battlefield scenarios.

MANA is easy to use, and offers a straightfoward interface for setting battle parameters. It supports a number of well-known movement algorithms, but doesn't have any way of changing or adding to these algorithms. It is extensively used in the defence community and has an active user base. Like EINSTein, it has a large number of parameters, which can lead to "over-fitting" of the model. It lacks some features useful for CS, such as different

movement algorithms for different groups of agents, and more advanced mechanisms for structural formation. Further, it lacks features useful in CAS, such as genetic algorithms and neural networks. It provides only basic statistics (tallies of agents detected, agents killed, etc.). MANA supports basic communication through situational awareness.

MANA could be used for very basic BSABMs, but for analysing CS and CAS a better choice that requires no programming would be EINSTein.

4.3.4 MASON

MASON ("Multi-agent Simulator Of Neighbourhoods / Networks") is a general purpose ABM library, geared towards speed, and large batch runs of simulations [Luke et al. 2005]. MASON is Java-based , like Repast, and provides many of the same features as Repast, but is lacking some important features. This was a deliberate design choice because a pre-existing library was more than capable of filling the role. For example, the excellent JFreeChart graphing library is bundled with the extra libraries available on the MASON web site. There is an evolutionary computation package provided as a separate project by the same lab, ECJ and this, along with the JFreeChart library, is considered as part of MASON for the purposes of this review.

Primarily, the advantage of MASON is in speed, however it is faster than Repast by only a small amount, and for some models is slightly slower than Repast, as reported by Railsback, Lytinen & Jackson [2006]. The other advantage that MASON has also relates to batch runs, in that a simulation can be stopped on one machine, then copied to and restarted on another machine. MASON lacks some of the batch parameter file format support that Repast has. It is a little more difficult than Repast to set up GUI (graphical user interface) displays and to display agents, because this must be done in a more sophisticated (arguably better) way. It is also somewhat difficult to draw icons (other than the geometric vector shapes provided) for agents on screen. It provides good inspection of agents' states on screen. Where MASON excels (through the ECJ library) is in adaptation—many types of evolutionary algorithms are supported, along with different types of agent learning. The network libraries combined with the flexibility afforded through virtue of being a Java library would make implementing networked causality easy.

MASON should be used where speed and/or sophisticated batch runs are required. MASON is also an excellent choice for exploring adaptation (both evolution and learning) as well as networked causality, and thus stands out as a clear winner in this report.

4.3.5 NetLogo

The name NetLogo comes from "Network Logo" and is a functional programming language [Finkel 1996] with "turtles" that represent the agents and have some state (unique identity, position, and user-defined attributes) [Wilensky 1999]. It is an easy-to-use platform and language that is slower than Repast and also difficult to extend. Nonetheless its ease of use, and support for automatic drawing of agents in 2D or 3D[2] makes this a suitable platform for beginner programmers.

[2]Only in the 3D preview version, which is very stable and the 3D functionality is easy to use.

In practical terms, the fact that it is a functional programming language means that it is relatively easy to learn and use. It also means that some programming language statements almost read as sentences, and, in combination with the set of commands that NetLogo provides, makes certain algorithms (such as finding all neighbours in a region) easy to write. However, the lack of true object-oriented features sometimes makes things difficult. For example, it can be unclear which methods are associated with which agents. Sometimes the actions of one class of agent may be written as part of a list of actions run by another class of agents. Though there is no explicit support for structure formation, the properties of the language itself as well as some specific built-in constructs enable effective feedback between agents, groups of agents, and the environment; thus NetLogo is a good environment for exploring structure formation. It is awkward to implement networked communications between agents, especially of multiple variables, even though NetLogo supports links between agents. NetLogo provides reasonably good ways of batching runs over a parameter space. It is not easy to extend some simple NetLogo models. For example, adding GIS data requires difficult programming statements and creative use of NetLogo features. The documentation and number of example models for NetLogo are both excellent, as is the user community, which provides a lot of support to new users of NetLogo.

4.3.6 Repast

Repast (REcursive Porous Agent Simulation Toolkit) is a well-established platform with many advanced features [North, Collier & Vos 2006]. It started as a Java implementation of the Swarm toolkit, but rapidly expanded to provide a very full featured toolkit for ABMs. Although full use of the toolkit requires Java programming skills, the RepastPy facility (and the features in the new Repast Symphony platform) mean that simple models can be implemented with little programming experience.

Repast is easy to extend to large, complicated models. It provides a wealth of useful facilities from 2D (and in the new Repast Symphony, 3D) displays, to charting models, logging results, and statistical packages. The current Repast stable includes the Colt Java libraries for statistics, the newer Repast Symphony includes the R statistical suite. This section describes the Repast stable library, because (as of writing) Repast Symphony was still feature incomplete.

One notable feature of the Repast library is its methods for reading input parameters, with specific features for batch runs, and support of two formats: custom plain text files or XML files using a custom schema. Repast also has a module for creating neural networks, which can be useful in implementing agent learning, and there are a few libraries useful in genetic algorithms [Fogel 2000, Fogel 2005], however there is no serious support for either. The documentation is good overall, and the user community is good, although some of the discussions are a little more technical than those on the NetLogo list, due to the advanced features that the use of Java can provide (such as threading the program to make sections run in parallel, and scheduling). Some of the API needs significantly better documentation, but overall it is good. As with MASON, the network libraries combined with the flexibility of being a Java library would make implementing networked causality easy.

4.3.7 Swarm

Swarm was originally developed at the Santa Fe Institute [Minar et al. 1996] and now is developed by the Swarm Development Group. It is the "ancestor" of many of the current ABM platforms. The basic architecture of Swarm is the simulation of collections of concurrently interacting agents, and this paradigm is extended into the coding, including agent inspector actions as part of the set of agents. So in order to inspect one agent on the display, you must use another hidden, non-interacting agent.

Swarm is a stable platform, and seems particularly suited to hierarchical models. As such, it supports good mechanisms for structure formation, through the use of multi-level feedback between agents, groups of agents, and the environment (all treated as agents). The Objective C Swarm requires learning Objective C, which can be a difficult language for inexperienced programmers. Objective C makes writing networked causality a little easier due to its message-passing style, however there is no direct library support for networking agents. There is also no direct support for mechanisms for adaptation. SWARM does provide some good statistical measures, however. The documentation seems a little terse in the API, but is generally good, and has a simple yet effective worked example. Some of the links on the Swarm Wiki are currently broken or point to the wrong area. Objective C Swarm was difficult to install on Mac OS X, and difficult to set up on Windows. To make effective use of software libraries, and to effectively manage, write, and debug software, an easy-to-use integrated development environment (IDE) should be used. Although the Emacs software can be used (and is recommended by the Swarm Development Group) as an IDE on all platforms, it is difficult to set up as such. Mac OS X does have an IDE (XCode) which supports Objective C. The Java version of Swarm feels cumbersome, and is worse than the Objective C Swarm in terms of documentation and code examples.

Swarm should be considered if the programmer has advanced skills, and if hierarchical models or structure formation are of interest.

4.3.8 WISDOM-II

WISDOM is the "Warfare Intelligent System for Dynamic Optimization of Missions" and contains a number of useful features in studying CAS in battlefield scenarios including C2, C3 and NCW concepts, as well as novel hierarchical structures at the battle-group level and also on the battlefield layout [Yang et al. 2006].

WISDOM-II has a number of innovative features such as a comprehensive NCW system, and good support for C2 and C3 agents. Its use of heirarchies in the battle-group and battlefield areas is very good and useful in modelling large groups of agents, and supports structure formation through multilevel feedback. It is capable of reasoning about behavours (at a group level) and of reporting of these and their effects in plain English. WISDOM-II has inbuilt non-linear C2 decision making, and so supports advanced networked causality, although the particular algorithm is fixed, along with the movement algorithms which are usually fixed (or limited) in BSABMs. It has a basic evolutionary algorithm, but again this is fixed. WISDOM-II unfortunately has poor documentation. It also required scenarios to be hand-coded in a complicated XML schema, with no way of debugging the file.

While WISDOM-II has many innovative features, the way in which simulations must be set up combined with the lack of error reporting, was found to make the tool difficult to use.

4.4 Previous analyses

Railsback, Lytinen & Jackson [2006] also developed a simple ABM and implemented it using a variety of different platforms. Their findings can be summarised as:

- MASON is a good choice for experienced programmers who are working on computationally intensive models.
- NetLogo is easy to use, and has excellent documentation. It is a good choice for models which are not too complicated, and where simulation time is not an issue. It is also a good choice for prototyping models that can later be extended and implemented in Repast (for example).
- Repast is the most complete platform, and has a comparatively good execution time of simulations. It has poor documentation in places, and also some "average" software engineering design choices, but is overall the best ABM platform.
- Swarm (Objective-C version) is stable, small, and well-organised and can support very complex models, but suffers from some of the faults of Objective C, including a lack of novice-friendly development tools and garbage collection.

The paper by Castle & Crooks [2006] reviewed ABM platforms including MASON, Repast and StarLogo (a variant on Logo, similar to the NetLogo platform reviewed herein), within a geographic information science (GIS) context. However, many of their findings apply more generally:

- MASON has a lack of documentation and has a relatively small user community. It lacked specific functionality that the authors were seeking, in terms of GIS integration. It also lacked other, more general functionality, specifically charting, although as noted there is a good charting package available from the MASON web site.
- Repast provides a lot of tools, and has good documentation, good examples and a large, active user community.
- StarLogo is simple to use, but has constrained functionality. The authors raise legitimate concerns about StarLogo and other non-Open Source platforms (which as of writing includes NetLogo) about the hidden nature of algorithms, lack of serious ability to extend, and possible non-repeatability of scientific (modelling) experiments. The authors also point out that in non-object oriented programming languages, such as NetLogo and StarLogo, which are functional programming languages, that there is not a clear mapping between agents in the conceptual framework and in the code, as there is with object oriented languages.

4.5 Comparison with previous analyses

As can be seen below, the experience of this author with the platforms was roughly in line with Railsback, Lytinen & Jackson [2006], however MASON is recommended over Repast due to criteria focussed on CS and CAS modelling requirements, rather than GIS integration. NetLogo also scored highly, and as per Railsback, Lytinen & Jackson [2006] this easy to use so would be appropriate for where users have limited programming experience. The previous subsections give more detail on each of the platforms evaluated. Note that some examples of Java and NetLogo code can be found in Appendix A, though these are for illustration purposes only and are not code for SimpleArmy.

5 Conclusions and Future Work

Ultimately, the choice of platform depends on the skillset of the person assigned to the task, and the particular details of the task. Some tasks may be possible in some of the BSABM platforms without programming. Other tasks may require agent behaviours that aren't available in fixed algorithm platforms, in this case BactoWars or one of the general ABM platforms would have to be used.

MASON and Repast are clear winners and are strongly recommended as an ABM platform. However, MASON requires a modelling with strong programming experience. For those with less programming experience, or for those who aren't interested in networked causality, NetLogo is highly recommended. For those who don't want to program and are interested in the battlefield scenario it models, EINSTein is a suitable choice.

In future work, a refined set of criteria will be established for making assessments. Assessments by a group of people will be canvassed to make the results more rigourous. Many of the software platforms are still under active development so it is also important to revise this document in the light of future changes to the platforms.

Acknowledgements

Feedback from Alex Ryan, Shane Magrath, Darryn Reid, Brandon Pincombe, Neville Curtis, Dion Grieger, Vanja Radenovic, Martin Wong, Axel Bender, and Victor Fok is greatly appreciated.

Appendix A: Software Code

A.1 Java

```
public class Agent {

        private int x, y; // position

        private int id;

        public Agent(int id) {
                x = 0;
                y = 0;
                this.id = id;
        }

        public void step() {
                x++;
                y++;
        }
}
```

A.2 NetLogo

```
turtles-own [
        id
]

to setup
        crt 10
        ask turtles [
                set id 1
                rt 45
        ]
end
to step
        ask turtles [
                fd 1
        ]
end
```

A.3 Objective C

(Code taken from Swarm documentation).

```
@interface Heatbug: SwarmObject {
double unhappiness; // my current unhappiness
int x, y; // my spatial coordinates
HeatValue idealTemperature; // my ideal temperature
HeatValue outputHeat; // how much heat I put out
float randomMoveProbability; // chance of moving randomly
Grid2d * world; // the world I live in
int worldXSize, worldYSize; // how big that world is
HeatSpace * heat; // the heat for the world
Color bugColor; // my colour (display)
}
-setWorld: (Grid2d *) w Heat: (HeatSpace *) h; // which world
    are we in?
-createEnd;
-(double) getUnhappiness;
-setIdealTemperature: (HeatValue) i;
-setOutputHeat: (HeatValue) o;
-setRandomMoveProbability: (float) p;
-setX: (int) x Y: (int) y; // bug's position
-setBugColor: (Color) c; // bug's colour (display)
-step;
-drawSelfOn: (id <Raster>) r;
```

References

Athale, C. & Deisboeck, T. S. (2006) The effects of egf-receptor density on multiscale tumor growth patterns, *Journal of Theoretical Biology* **238**(4), 771–779.

Athale, C., Mansury, Y. & Deisboeck, T. S. (2005) Simulating the impact of a molecular 'decision-process' on cellular phenotype and multicellular patterns in brain tumors, *Journal of Theoretical Biology* **233**(4), 469–481.

Castle, C. J. E. & Crooks, A. T. (2006) *Principles and Concepts of Agent-Based Modelling for Developing Geospatial Simulations*, Technical Report 110, Centre for Advanced Spatial Analysis, University College London, UK.

Epstein, J. & Axtell, R. (1996) *Growing Artificial Societies. Social Science from the Bottom Up*, MIT Press.

Finkel, R. A. (1996) *Advanced Programming Language Design*, Addison-Wesley. URL – `ftp://ftp.aw.com/cseng/authors/finkel/apld/`.

Fogel, D. B. (2000) What is evolutionary computation, *IEEE Spectrum* **37**(2).

Fogel, D. B. (2005) *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEEE Press Series on Computational Intelligence, third edn, Wiley-IEEE Press. URL – `http://www.amazon.co.uk/exec/obidos/ASIN/0471669512/citeulike-21`.

Gardner, M. (1970) The fantastic combinations of John Conway's new solitaire game "life", *Scientific American* **223**, 120–123.

Grimm, V., Revilla, E., Berger, U., Jeltsch, F., Mooij, W. M., Railsback, S. F., Thulke, H. H., Weiner, J., Wiegand, T. & Deangelis, D. L. (2005) Pattern-oriented modeling of agent-based complex systems: lessons from ecology, *Science* **310**, 987–991.

Ilachinski, A. (1999) Towards a science of experimental complexity: An artificial-life approach to modeling warfare, *in 5th Experimental Chaos Conference*.

Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K. & Balan, G. (2005) Mason: A multiagent simulation environment, *Simulation* **81**(7), 517–527. URL – `http://dx.doi.org/10.1177/0037549705058073`.

Minar, N., Burkhart, R., Langton, C. & Askenazi, M. (1996) *The Swarm simulation system: a toolkit for building multi-agent simulations.*, Technical report.

Najlis, R., Janssen, M. A. & Parker, D. C. (2001) Software tools and communication issues, *in Proc. Agent-Based Models of Land-Use and Land-Cover Change Workshop*, pp. 17–30.

Newth, D. & Finnigan, J. (2006) Emergence and self-organization in chemistry and biology, *Australian Journal of Chemistry* **59**, 841–848. URL – `http://dx.doi.org/10.1071/CH06292`.

North, M. J., Collier, N. T. & Vos, J. R. (2006) Experiences creating three implementations of the repast agent modeling toolkit, *ACM Trans. Model. Comput. Simul.* **16**(1), 1–25. URL – http://portal.acm.org/citation.cfm?id=1122012.1122013.

Railsback, S. F., Lytinen, S. L. & Jackson, S. K. (2006) Agent-based simulation platforms: Review and development recommendations, *Simulation* **82**, 609–623.

Reynolds, C. W. (1987) Flocks, herds and schools: A distributed behavioral model, *SIGGRAPH Comput. Graph.* **21**(4), 25–34. URL – http://dx.doi.org/10.1145/37402.37406.

Ryan, A. (2007) *A Multidisciplinary Approach to Complex Systems Design*, PhD thesis, The University of Adelaide.

Tesfatsion, L. (2003) Agent-based computational economics: modeling economies as complex adaptive systems, *Information Sciences* **149**, 263–269.

Theraulaz, G. & Bonbeau, E. (1999) A brief history of stigmergy, *Artif. Life* **5**(2), 97–116. URL – http://dx.doi.org/10.1162/106454699568700.

Wilensky, U. (1999) Netlogo. URL – http://ccl.northwestern.edu/netlogo/.

Yang, A., Curtis, N., Abbass, H. A., Sarker, R. & Barlow, M. (2006) *WISDOM-II: A Network Centric Model for Warfare*, ANU E Press, chapter 8, pp. 149–173. URL – http://epress.anu.edu.au/cs/pdf/ch08.pdf.

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
DOCUMENT CONTROL DATA

1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)

2. TITLE

Review of Software Platforms for Agent Based Models

3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION)

Document	(U)
Title	(U)
Abstract	(U)

4. AUTHOR

Matthew Berryman

5. CORPORATE AUTHOR

DSTO Defence Science and Technology Organisation
PO Box 1500
Edinburgh South Australia 5111 Australia

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO-GD-0532	AR 014-164	General Document	April 2008

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. NO. OF PAGES	12. NO. OF REFERENCES
2008/1003280			23	33

13. URL on the World Wide Web

http://www.dsto.defence.gov.au/corporate/reports/DSTO-TR-0532.pdf

14. RELEASE AUTHORITY

Chief, Land Operations Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SA 5111

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CITATION IN OTHER DOCUMENTS Yes

18. DSTO RESEARCH LIBRARY THESAURUS

Modelling
Agent based distillations
Complex systems
Software evaluation

19. ABSTRACT

This report reviews both general agent based modelling (ABM) and battlefield-specific ABM toolkits against established criteria, namely flexibility, documentation, speed, and facilities. The other main focus of this report is to consider which ABM toolkits are best suited to studying self-organisation, adaptation, and causality in networks, since these are all essential parts of complex adaptive systems of interest to defence. The ABM toolkits evaluated against these requirements were BactoWars, EINSTein, MANA, MASON, NetLogo, Repast, Swarm and WISDOM-II.